International Journal of Database Theory and Application
Vol.9, No.4 (2016), pp.277-288
http://dx.doi.org/10.14257/ijdta.2016.9.4.26

Design of Query Reformulation Engine in Data Access and Integration System

Xiyin Liu*, Lijun Cao and Zhongping Zhang

Hebei Normal University of Science & Technology, China
College of Information Science and Engineering Yanshan University, China
liuxiyin2003@sina.com

Abstract

This paper introduces three core modules of query reformulation engine, mapping document, query reformulation module, and statement conversion module. Mapping document is an XML document that keeps the mapping information between local data source and related data sources; using mapping document, applications could find data sources that have mapping relationship with its local data source. The query reformulation module reformulates the query statements submitted by users to local data resource to query statements to all data sources that have mapping relationship with local data resource. The statement conversion module converts XPath statements to OQL statements that are supported by OGSA-DQP; through OGSA-EDAI in the bottom layer, the access result to the data sources could be obtained. When a user submits an XPath statement to OGSA-DQP, it calls the query reformulation module, which first checks the mapping document to find information of other data sources, then expands and reformulates this XPath statement into query statements that are suitable for the mapped data sources. Afterward, the statement conversion module converts the reformulated XPath statements into OQL statements and returns to OGSA-DQP, which then performs the query operation.

Keywords*: integrated, reformulation, schema mapping, query engine*

1. Introduction

For schema mapping, traditionally there are three approaches [1-4]. The first one is GAV (Global As View), which requires global schemas be represented as local schemas; the second one is LAV (Local As View), which requires global schemas be represented independent to local schemas, for which mappings from each local schema to the global schema need to be established; the third one is GLAS, a integration of GAV and LAV, which is still in its early stage of development. [5-9]

With the development of grid environment, its high-dynamic and distributed features demand more than the centralized structure of conventional data integration systems. In grid environments, adding relative data sources into the system requires rewritten of the global schema, and the repeating of such process for each addition of data source is laborious. Moreover, global schema is a bottleneck of the system, for that if the global schema server collapses, the whole system falls apart with it.

This paper made developments and innovations in the following aspects:

In this paper, we present an incompact method, where mapping rules are established directly among data source schemas instead of on a centralized regulator or a multi-regulator system. Instead of using single data schema to represent all data sources in global mode, this method employs a series of local schemas. When a new node (or data source) is being added to the system, it only needs to provide a mapping rule set, which describes its own schema and the schemas of other data sources linked with it.This approach brings flexibility and reliability, for that a node could be added or removed

ISSN: 2005-4270 IJDTA

dynamically, which fits the high-dynamic activities in grid environment; and also, the schemas of each data source are only directly connected to a limited number of other data source schemas, and if the schemas it is not connected to are inside its transitive closure, these schemas are still reachable. We also designed a query reformulation algorithm to solve the problem of the query reformulation engine.

2. Schema Mapping

The primary purpose of the query engine is to establish an incompact semantic network, forming queries to distributed, heterogeneous data sources through semantic bindings; the underlying network is composed by a series of autonomous nodes which contain relevant data, and the data sources are connected through the establishment of mappings; the schema mappings of the data sources are stored in rule documents.

2.1. Establishment and Rules of Schema Mapping

Conventional schema mapping is often accompanied by schema conflicts, including naming conflict, list structure conflict, data conflict, attribute conflict, and semantic conflict. In this system, the establishment of mapping between two data sources faces conflicts too. We presented in this paper a new solution to these conflicts.

In this study, each node of the system is treated as a peer point and linked by establishing mapping relationships among data sources, thus unconventional mapping approach is employed. The structural heterogeneity of the XML data source is compensated by combining the paths of different schemas. Mappings are illustrated as path expressions, and in the mappings contained an explicit element or attribute of the data source schema and the elements and attributes of the related target data source schemas. The data integration model proposed in this paper employs XPath query statements to express the path-to-path mappings, while assuming the data of all data sources uses XML schema as data model, *i.e.* this model uses a series of XPath expressions to describe the path of a data source.

As the first step of establishing schema mappings, a small subset of XPath language needs to be considered. This subset is expressed by:

$$q \rightarrow n \mid .\mid q/q \mid q // q \mid q\,[q] \quad (1)$$

where n represents a random label, "." is the current node, "/" is the child node axis, "//" is the offspring node axis, and "[]" represents a predicate.

A schema mapping is defined as a group of "Formulas", and the two sides of the formula represent a pair of schemas. In particular, for a data source schema S a mapping M is defined with a set of mapping

Rules $\Re^M = \{R_1^M, R_2^M, ...R_k^M\}$. When the execution of a path reaches its mappings, each mapping rule links to a path in different schemas. A mapping rule could be expressed in the form of Expression (2).

$$R^M : \{S^S, P_S\} \rightarrow C^M \{S_D, P_D\} \quad (2)$$

Where R^M is the label of the rule, S_S is the data source schema related to the establishment of the rule, P_S is the path expression of the data source schema, S_D is the target data source schema related to the establishment of the semantic binding, P_D is the path expression of the target data source schema, and C^M is the element of the radix representing the mapping between two related schemas. Mappings could be identified as 1-1, 1-N, N-1, and N-N modes, determining by the number of nodes of the schemas with mappings established.

2.2. Schema Mapping Document

Schema mapping rules are stored in documents called schema mapping documents. Each data source involved in grid data accessing and integrated system stores and maintains a corresponding schema mapping document, in which the information of all the mapping rules related to this node is stored.

Figure 1 presents the structure of schema mapping documents. It can be seen that the expressions of the elements of a single data source schema and a series of elements defining the mapping rules are included in the document. The rule elements contain a complex structure, which requires clear illustration of all the paths and radix constraints of the mappings.

```
<schema targetNamespace="http://SHM/SHMDocument"
        xmlns="http://www.w3.org/2001/XMLSchema"?>
  <element name="Mapping">
    <complexType>
      <sequence>
        <element name="sourceSchema" type="string"
                 minOccurs="1" maxOccurs="1"/>
        <element name="Rule" minOccurs="1">
          <complexType>
            <sequence>
              <attribute name="Cardinality" type="string"
                         minOccurs="1" maxOccurs="1"/>
              <element name="sourcePath" type="string" minOccurs="1"/>
              <element name="destSchema" type="string"
                         minOccurs="1" maxOccurs="1"/>
              <element name="destPath" type="string" minOccurs="1"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

Figure 1. XML Schema of Schema Mapping Document

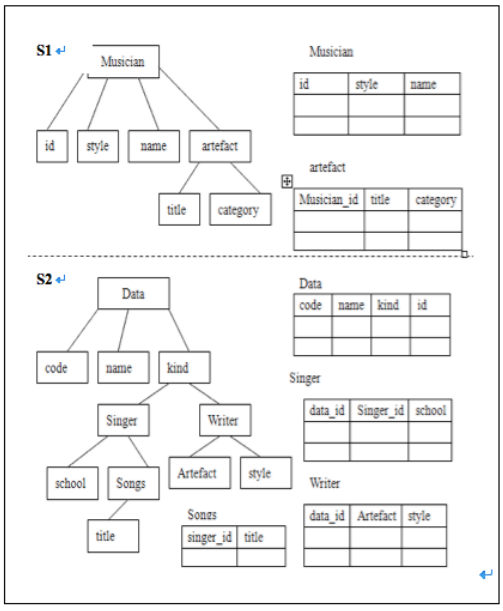

Figure 2. Example of Database Schema

An illustrative example is presented in Figure 2. The underlying databases are relational databases, but their schemas could be expressed with XML and handled with query reformulation algorithms. Each of the two nodes contains a dependent and autonomous database which stores the information of musicians and their works. Two self-explanatory views are presented in the figure, one hierarchical (representing the local XML database), the other graphical (representing objects or relational databases).

In OGSA-DQP, the table schemas of a database is described or retrieved in the form of XML documents.

The table schema of the sample database is presented below in the form of XML document.

In OGSA-DQP, table schemas of a database could be given or retrieved in the form of XML documents. When schema mapping between data sources needs to be established, this mapping should be able to reflect the data domain semantic bindings of the different data sources, including primary and foreign keys. In this study, the form presented in Figure 3 is proposed for the establishment of mapping.

(1) S1/Musician/style-> S2/Singer/school, S2/Writer/style
(2) S1/Artefact/title-> S2/Songs/title, S2/Writer/artefact
(3) S1/Musician/id-> S2/Data/id
(4) S1/Artefact/Musician_id-> S2/Singer/data_id, S2/Writer/data_id

Figure 3. Establish Schema Mapping

Thereby, the schema mappings among data sources are established, and these mappings should be saved to the schema mapping document of the local node. A segment of the schema mapping document of node S1 is extracted as shown in Figure 4.

```
<Mapping>
 <sourceSchema>S1</sourceSchema>
  <Rule cardinality="Mapping1-N">
   <destinationSchema>S2</destinationSchema>
   <sourcePath>/Musician/style</sourcePath>
   <destinationPath>/Data/kind/Singer/School </destinationPath>
   <destinationPath>/Data/kind/Writer/artifact</destinationPath>
 </Rule>....
</Mapping>
```

Figure 4. Schema Mapping Document Fragment of S1

This way, the mappings stored in the local schema mapping document are realized. Through the establishment of schema mappings, semantic bindings are built for the data sources, founding the basis of the query reformulation algorithm of the next step.

3. Query Reformulation

The core of the query reformulation module is the query reformulation algorithm. Having discussed in Chapter 2, a schema mapping document is stored at each node, and this document contains the mapping rules between the local node schema and other node schemas, among which schema mappings are established and mutual semantic bindings are built. Users submit an XPath query statement to local schema, and the task of the query reformulation algorithm is to reformulate this statement.

By executing the query reformulation algorithm prior to the execution of query statements, the semantic bindings established on each node of the system are obtained, and more information of other data sources are made available. With this approach, when

a query statement is submitted to a data source schema, the system would obtain or use the data on any data source so long as this data source could be reached through semantic mappings. The system will reformulate the statement, extend and convert it into query statements that are suitable for each semantic related data source. Therefore, by simply giving an XPath query statement to local data source schema, a user could retrieve data from all the linked data sources in the system.

With one XPath statement input and mappings, the query reformulation algorithm will produce zero, one, or several reformulated query statements. Let Q denote the input XPath statement, S denote the data source schema where Q is submitted, M denote the mappings in the system, and Q_{R_i} denote the reformulated query statements. The logic of the algorithm will be detailed below in five major steps.

(1). Identify the path expressions of Q. One or more predicates may be found in every XPath query statement, and these predicates could generate different branching points in a tree structure and bring more query statements. Each one of these branches identifies an explicit path in the XML data source, and in the query statement the identified paths are collected into a set P.

(2). Search for the candidate paths of all data source schemas related to S. The primary purpose of this step is to find the corresponding paths of all the data sources that are semantic related to S. Such purpose means to use the information in the schema mapping document of S and search for the corresponding path expression of each element Pi in set P. To be more accurate, for each Pi in query statement Q, the algorithm searches for all the corresponding paths in the schemas, while these schemas transitively connect to S through the path Pi. Paths like $P^{\diamond}_{i,j}$ are called candidate paths, and the schemas contain them $S^{\diamond}_{j}$ are called candidate schemas. Meanwhile, a candidate unit $E^{\diamond}_{i,j}$ is also defined, and it is given in the form of $\langle S^{\diamond}_{j}, \{P^{\diamond}_{i,j}\}\rangle$, where $P^{\diamond}_{i,j}$ is a path set on schema $S^{\diamond}_{j}$. Therefore, for each path expression Pi∈P, with 0<j<n (n is the number of data source schemas in the system), zero, one or more candidate elements $E^{\diamond}_{i,j}$ are generated. The candidate set $\varepsilon^{\diamond}$ is a set of candidate elements $\{E^{\diamond}_{1}, E^{\diamond}_{2}, ..E^{\diamond}_{n}\}$, where $E^{\diamond}_{j} = \cup_i E^{\diamond}_{i,j}$. Then, this step returns a set $\varepsilon^{\diamond}$ as the result.

(3). Trim the candidate schemas. In the last step, the candidate schemas are found, and in this step, each candidate schema needs to be verified to see if it could be used to obtain one or more reformulated statements of query statement Q. To achieve this purpose, the query reformulation algorithm checks each candidate schema to see for each path expression of the statement whether it has at least one candidate path. Also, the algorithm verifies that none of the candidate paths has been used to reformulate Q, so that extra consideration of the paths can be avoid. The schemas survived the verification are considered in the reformulation, and they are called target schemas. A target unit $E^{*}_{i,j}$ is also defined in the form of $\langle S^{*}_{j}, \{P^{*}_{i,j}\}\rangle$, where $P^{*}_{i,j}$ is a path set on S^{*}_{j}. Thereby, for each path expression Pi in P, zero, one, or more candidate elements are generated. The candidate set ε^{*} is a set of candidate elements $\{E^{*}_{1}, E^{*}_{2}, ..E^{*}_{n}\}$, where $E^{*}_{j} = \cup_i E^{*}_{i,j}$. Then, this step returns a collection ε^{*} as the result.

(4). Reformulate query statements. In this step, for the returned set ε^{*}, the algorithm will generate one or more XPath query statements on the basis of each of the schemas in the set. Detailing below is the process carried out for each target schema S^{*}_{j} in ε^{*}:

a. Estimate radix constraints. If in collection P, each path Pi has a corresponding path on schema S^{*}_{j}, then all the mappings between S and S^{*}_{j} are of the type 1-1 or N-1. Then the output reformulated query statements are just single statement expressed on target schema S^{*}_{j}. Contrarily, if in P multiple target paths on schema S^{*}_{j} exist, then multiple reformulated query statements will be generated on schema S^{*}_{j}. The number of the reformulated query statements depends on the possible combinations of paths. If the radix

of set P is k, and denoted by |P|, then for schema S_j^*, the number of path combinations com equals to the product of the radixes of each path , *i.e.* $com = |P_{1,j}^*| \times |P_{2,j}^*| \times ..... \times |P_{k,j}^*|$.

b. Check the binding conditions. Once the mapping radixes are established, before the actual generation of reformulated statements, the binding conditions among the paths $P_{i,j}^*$ of target schemas S_j^* need to be checked. In the 1-1 and N-1 mapping cases, since there will only be a single reformulated query statement generated, if in schema S_j^* at least two paths exist and the constraints are not considered, no reformulated statement will be generated. Also, in the 1-N mapping case, as long as each path combination fulfills the binding condition, equivalent number of reformulated statements will be generated.

c. Construct XPath query statements. After checking the binding conditions of the target paths, the actual construction of one or more XPath query statements is initialized. Thereby, the reformulated query statements of the query statement Q are the query statements to target schema S_j^*.

(5). Call the query reformulation algorithm circularly. Circularly call the query reformulation algorithm to the reformulated statements in order to obtain the query statements corresponding to every transitive mapping.

Combining the information of the two data sources given above, following is an illustrative example of the work of query reformulation engine. corresponding to the operations shown in Figure 5 and 6. After checking the XML schema expression of local data source, a user submits an XPath query statement Q=/Musician[Style=“Pop”]/artefact/title to S1 to retrieve the names of the works of the musicians styling in “pop”. After accepting the initial query statement the query reformulation engine works by the following procedure. First, it identifies the paths P1 and P2 in Q and generates an output set P, and then it retrieves the schema mapping document stored and maintained in schema S1, and in this document stored the schema mapping rules of S1 with other related data sources. The algorithm then finds two mapping rules to schema S1 and S2 from P1 and P2 respectively. Precisely, one of the rules relates P1 to the two paths in schema S2, and the two paths are /Data/kind/Singer/school and /Data/kind/Writer/style, while the other rule relates P2 to /Data/kind/singer/songs/title and /Data/kind/writer/artefact. Therefore, in step 2 the algorithm returns a candidate set composed by element $P_{i,j}^{\Diamond}$ and schema $S_2^{\Diamond}$. In this example, since schema $S_2^{\Diamond}$ is related to both path P1 and P2, it is identified as a target schema and used to reformulate the query statement Q. Eventually, the algorithm produces two query statements based on schema S2, named Q_{R_1} and $Q_{R_{21}}$, which will be executed on S2.

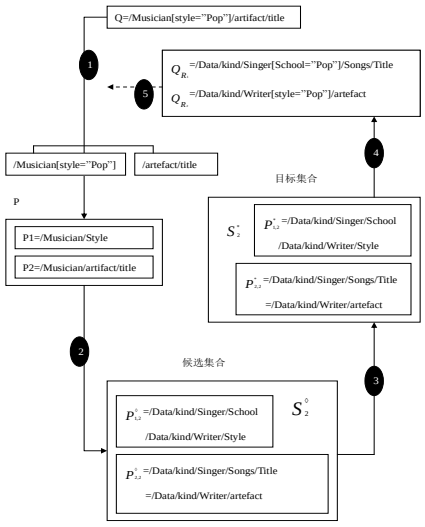

Figure 5. Example of Query Reformulation Algorithm

Following are the pseudo codes of the query reformulation algorithm.

Algorithm 5.1: Query Reformulation Algorithm

Input: Query statement Q, schema S, and mapping M (mapping rule that is saved in the schema mapping document)

Output: Reformulated query statement set B^*

QRA (Q,S,M)

Begin

(1) Identify the paths of query statement Q, and assign to the set P;

(2) for each path Pi in P do

(3) look for candidate path through Pi and M, and assign to $\varepsilon^{\Diamond}$;

(4) trim candidate schema $\varepsilon^{\Diamond}$, and assign to ε^*;

(5) for each S_j^* in ε^* do

(6) if (S and S_j^* are 1-N mappings) then

(7) merge paths $E_{i,j}^*$, and assign to $B^{\Diamond}$;

(8) for each candidate query statement $Q^{\Diamond}$ in $B^{\Diamond}$ do

(9) if (check if $Q^{\Diamond}$ satisfies the connection condition) then

(10) assemble $Q^{\Diamond}$ into query statement, and assign to Q^*;

(11) $B^{rec} \leftarrow$QRA(mapping documents of Q^*, S_j^*, and S_j^*);

(12) if (the radix of B^{rec} is greater than 0) then

(13) $B^* \leftarrow B^{rec} \cup B^*$;

(14) $B^* \leftarrow B^* \cup Q^*$;

(15) else

(16) if(check if $E_{i,j}^*$ satisfies the connection condition) then

(17) assemble $Q^{\Diamond}$ into query statement, and assign to Q^*;

(18) $B^{rec} \leftarrow$QRA (mapping documents of Q^*, S_j^*, and S_j^*);

(19) if (the radix of B^{rec} is greater than 0) then

(20) $B^* \leftarrow B^{rec} \cup B^*$;

(21) $B^* \leftarrow B^* \cup Q^*$;

(22) return B^*

End

4. Statement Conversion

The query statement submitted by user and the query reformulation statements handled by the reformulation engine are all in the form of XPath statements. However, the underlying distributed query processer OGSA-DQP supports OQL statements only, thus a statement conversion module became necessary. The XPath statements given with the XML expressions of the relational databases supported by OGSA-DQP are mostly in the form of Expression (3).

/database_A[predicateA]/table_A[predicate_B]/column_A (3)

Where, predicate_A ::= table_pred_A[column_pred_A = value_pred_A], and Predicate_B ::= column_pred_B = value_pred_B。

Thereby, the mappings to the "select", "from", and "where" sub-statements of OQL are obvious. column_A defines the "select" attribute, while table_A and table_pred_A define the "from" sub-statement. If the column_pred_A = value_pred_A, and column_pred_B = value_pred_B, then they should be put in the "where" domain.

In this system, there are actually two additional functions provided to OGSA-DQP.

(1) Query reformulation. When the user submits an XPath statement to the schema of local data source, the system will reformulate the statement and generate corresponding query statements to related data sources.

(2) Conversion of query language. The reformulated query statements are still XPath statements, while the distributed query processor only supports OQL statements. Therefore, we added in the system the function of converting the reformulated query statements into OQL statements in the query reformulation engine.

These two functions are implemented in SchemaMappingActivity, and the schema of the new activity is as shown in Figure 6. The element Expression contains the submitted XPath query statement and ServiceLocation contains the addresses of related Web services, and these addresses will generate the reformulated query statements later. This way, OGSA-DQP will support two types of operations as shown in Figure 7 OQL query statements, and XPath query statements, which actually will be converted into OQL statements after reformulation.

```
<activityConfiguration> ...
    <activityMap>
      <activity name="XPathMappingStatement"
                implementation="uk.org.ogsadai.dqp.gdqs.XPathMappingActivity"
                schema="xpath_mapping_statement.xsd"/>
      <activity name="oqlQueryStatement"
                implementation="uk.org.ogsadai.dqp.gdqs.OQLQueryStatementActivity"
                schema="oql_query_statement.xsd"/>
...
</activityMap>
</activityConfiguration>
```

Figure 6. Fragment of Activity Configuration Document of OGSA-DOP

```
<?xml version="1.0" encoding="UTF-8"?> ...
    <xsd:schema
      <xsd:complexType name="XPathMappingType">
        <xsd:complexContent>
          <xsd:extension base="gds:ActivityType">
            <xsd:sequence>
              <xsd:element name="expression"
                  minOccurs="1" maxOccurs="1">
                <xsd:complexType mixed="true">
                  <xsd:complexContent>
                    <xsd:extension base="gds:ActivityInputType"/>
                  </xsd:complexContent>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="ServiceLocation"
                  minOccurs="1" maxOccurs="1">
                <xsd:complexType mixed="true">
                  <xsd:complexContent>
                    <xsd:extension base="gds:ActivityInputType"/>
                  </xsd:complexContent>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="webRowSetStream"
                  minOccurs="1" maxOccurs="1">
                <xsd:complexType mixed="true">
                  <xsd:complexContent>
                    <xsd:extension base="gds:ActivityOutputType"/>
                  </xsd:complexContent>
                  </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    <xsd:element name="XPathMappingStatement"
          type="gds:XPathMappingType"
          substitutionGroup="gds:activity"/>
</xsd:schema
```

Figure 7. Schema of New Activity

5. Experiment and Related Analysis

A query reformulation engine in data access and integration system, which is service oriented and suitable for grid environment, has been presented in this paper. This section will introduce an experiment, which aims to verify the feasibility of the proposed query reformulation engine, and to analyze its time consumption and performance on the basis of feasibility.

5.1. Data and Preparation of the Experiment

The experimental data is from the DB Research data set, which is a set of data on papers, articles with time they were published. This paper selects three databases: ICDE, VLDB and SIGMOD to verify the feasibility of the proposed engine and to obtain some related analysis. The corresponding database XML representations of the three databases are shown in Figure 8 while these databases are respectively installed in three computers connected in local area network.

```
<databaseSchema dbname="ICDE">
    <table name="Journal">
      <column name="Artical"/>
      <column name="Year"/>
    </table>
  </databaseSchema>

  <databaseSchema dbname="VLDB">
     <table name="Proceedings">
       <column name="Paper"/>
        <column name="Year"/>
     </table>
  </databaseSchema>

<databaseSchema   dbname="SIGMOD">
   <table name="Paper">
     <column name="Title"/>
      <column name="Year"/>
    </table>
</databaseSchema>
```

Figure 8. XML Representation of Databases

Mapping rules are firstly needed to establish. The mapping rules of the mapping documents on node maintaining of ICDE database are /ICDE/journal/article->/VLDB/proceedings/pap and /ICDE/journal/year->/VLDB/proceedings/year. On the other hand, VLDB/proceedings/paper->/SIGMOD/paper/title and /VLDB/proceedings/year ->/SIGMOD/paper/year are the rules of the mapping documents on node maintaining of VLDB database.

5.2. Experiment Results

The experiment contains two parts. One part is to verify the feasibility while the other part is to analyze time consumption of the system.

After retrieving the mode of local data source, users give an XPath query code QICDE=/ICDE/journal[year= "2005"]/article to the nodes saved in ICDE database. In this manner, the system can not only get query results of local data source, but also obtain query results of the other two databases, which are shown in Figure 9.

In the measurement of the consumption of system time, there are five measurement parameters.

N: the number of waiting queries after extension of reformulation;

T1: time used by the query reconstruction engine from the initial query to return of the OQL statement after the reformulation;

T2: execution time of distributed processors after they receive the OQL query statement;

(4) T: time that users have to wait for;

(5) T3: other time, T3=T-T1-T2.

The experiment is conducted in two conditions, which call the data services locally and remotely. The time used by these two conditions is shown in Figure 10 and 11.

Figure 9. The Return Results of Given Queries

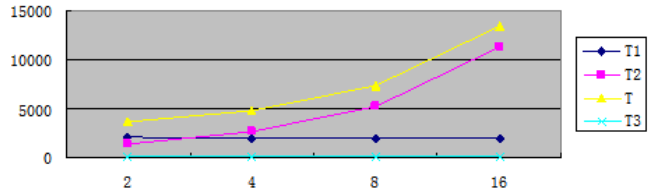

Figure 10. Response Time when Service Calls are Local

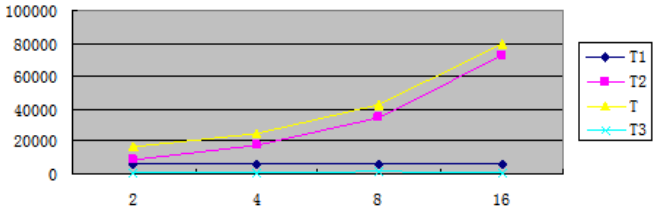

Figure 11. Response Time when the Services Calls are Remote

From the results in Figure 9, by establishing the mapping between the data sources, users can obtain the access results of the other two data sources after submitting just one query. This demonstrates the system is feasible.

In Figure 10, T2 increases rapidly when there are more queries, while T1 and T3 keep stable. In Figure 11, the total used time T is relatively large because it is the remote call service, while other conditions are similar with those in local call service. T2 also increases linearly, while T1 and T3 are stable.

Two conclusions can be obtained from the above observation. Firstly, the proposed query rewriting engine for the data access and integration system is feasible, and suitable for data access and integration in grid environment. Second, in most cases, the time used of the proposed engine is relatively small and quit stable, which is acceptable for users.

6. Conclusions

The schema mapping document in query reformulation engine stores the schema mapping rules. By establishing the relations among data sources with these rules, a correlative Peer network is formed, therefore the bottleneck of global schema is overcome.

The interaction of the engine is that when user submits an XPath statement to OGSA-DQP, it calls the query reformulation engine. Then, the engine first retrieves the schema mapping document, finds the information of related data sources, and then extends and reformulates the XPath statement to generate suitable query statements to other established related data sources. At last, through statement conversion module, the obtained reformulated XPath statements are converted to OQL statements supported by OGSA-DQP and return them to OGSA-DQP, which then execute the query operation and completes the database access.

References

[1] A. Cali, D. Calvaness and G. D. Gianomo, “On the expressive power of data integration Systems”, Berlin, Springer, **(2003)**, pp. 338-350.
[2] A. Y. Halevy, “Answering queries using views: a survey”, The International Journal on Very Large Data Bases, vol. 10, no. 4, **(2001)**, pp. 270-294.
[3] A. Y. Halevy, “Logic—based techniques in data integration”, In: Minker J. Logic Based Artificial Intelligence. New York: Kluwer Publishers. **(2000)**, pp. 575-595.
[4] M. Lenzerini, “Data Integration: a theroretical perspective”, In: Wisconsin M. Proceedings of Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. New York: ACM Press, **(2002)**, pp. 233-246.
[5] S. Castano and V. Antonellis, “Global viewing of heterogeneous data sources”, Transactions on Knowledge and Data Engineering, vol. 13, no. 2, **(2001)**, pp. 277-297.
[6] H. T. Mogulkoc, D. W. Coit and F. A. Felder, “Electric power system generation expansion plans considering the impact of Smart Grid technologies”, International Journal of Electrical Power & Energy Systems, vol. 42, no. 1, **(2012)**, pp. 229-239.
[7] A. Usman and S. H. Shami, “Evolution of Communication Technologies for Smart Grid applications”, Review Article. Renewable and Sustainable Energy Reviews, vol. 19, **(2013)**, pp. 191-199.
[8] A. P. Malozemoff, “New Material Requirements for Superconductor Grid Technology”, Original Research Article. Physics Procedia, vol. 36, **(2012)**, pp. 1429-1433.
[9] V. Giordano and G. Fulli, “A business case for Smart Grid technologies”, A systemic perspective Original Research Article. Energy Policy, vol. 40, **(2012)**, pp. 252-259.

